



## **Lab Midterm**

**Name: Fizza Bukhari**

**Registration number: CIIT/SP21/BCS-007/ATK**

**Course: Compiler Construction**

**Submitted to: Sir Bilal Haider Bukhari**

**Date: 05 April 2024**

### **Qusetion#01:**

#### **Briefly describe the regex library of C#**

In C#, Regular Expression is a pattern which is used to parse and check whether the given input text is matching with the given pattern or not. In C#, Regular Expressions are generally termed as C# Regex. In C#, Regular Expressions (regex) serve as powerful patterns utilized to parse and validate input text against predefined patterns, facilitating precise pattern matching within strings. The C# regex package, often referred to as C# Regex, encompasses a rich set of functionalities essential for string manipulation through pattern matching techniques. Situated within the 'System.Text.RegularExpressions' namespace, this library presents developers with pivotal classes like 'Regex', 'Match', 'Match Collection', and 'Group', each playing a distinct role in enabling various string processing tasks. The 'Regex' class acts as the primary interface for defining and applying regular expression patterns, while the 'Match' class provides detailed information about individual match results, including captured groups and their positions within the input text.

Furthermore, the C# regex library extends its capabilities through the 'Match Collection' class, enabling the storage and retrieval of multiple match results obtained during a single search operation. Additionally, the 'Group' class facilitates the examination and extraction of specific segments or sub-patterns within matched text, further enhancing the flexibility and precision of pattern matching tasks. Leveraging the conventional regex syntax, encompassing metacharacters, character classes, quantifiers, and more, C# regex empowers developers to perform a diverse array of operations on strings, including search and replace, input validation, and data extraction. This comprehensive toolkit stands as an indispensable asset for C# developers seeking to achieve efficient and reliable string manipulation within their applications, enabling them to handle complex text processing tasks with ease and accuracy.

### **Qusetion#02:**

#### **Make recursive descent or LL1 parser or recursive descent parser for the following grammar:**

**S -> X\$**

**X -> X % Y | Y**

**Y -> Y & Z | Z**

**Z -> k X k | g**

First, we remove left recursion from the given grammar. Modified grammar is given below;

**S -> X\$**

**X -> Y X'**

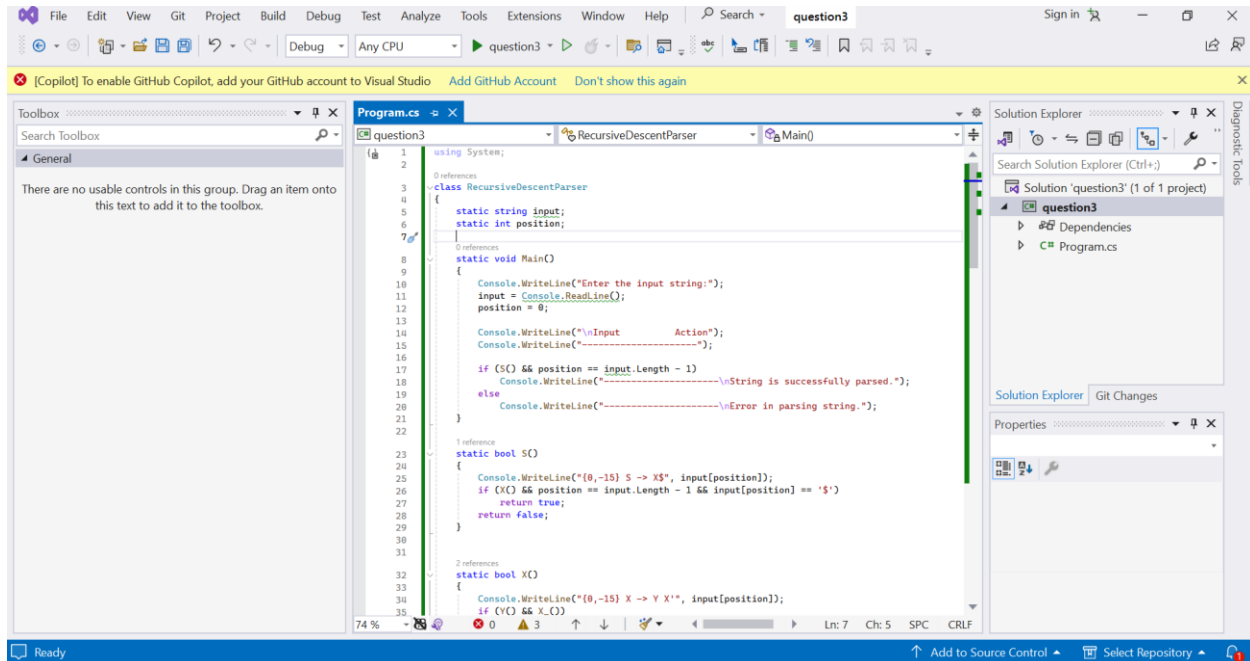
**X' -> % Y X' | ε**

**Y -> Z Y'**

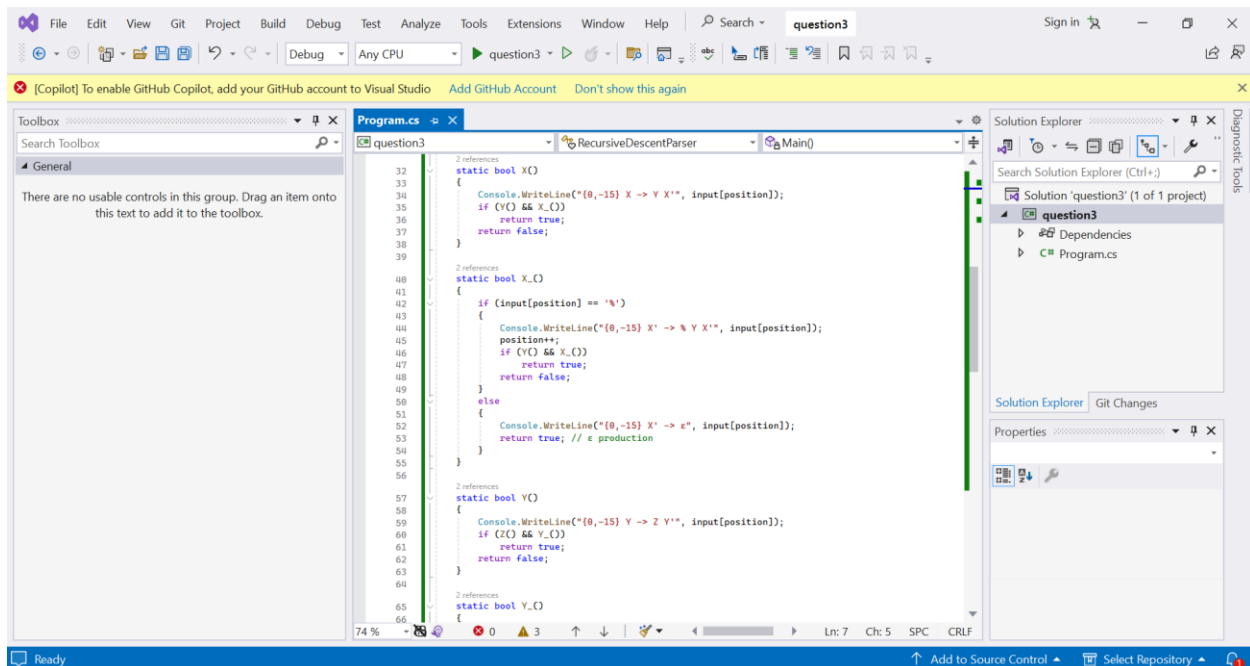
$Y' \rightarrow \& Z Y' \mid \epsilon$

$Z \rightarrow k X k \mid g$

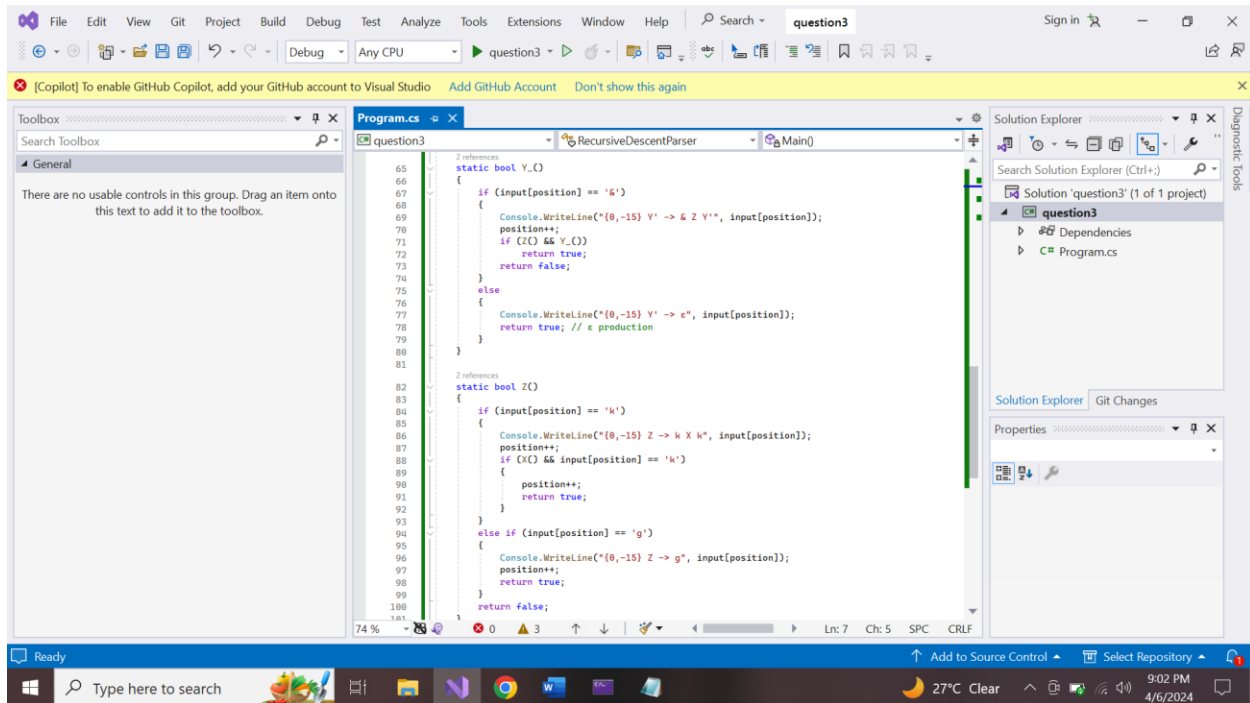
Now given below is recursive descent parser code in c#



```
1 using System;
2
3 class RecursiveDescentParser
4 {
5     static string input;
6     static int position;
7
8     static void Main()
9     {
10         Console.WriteLine("Enter the input string:");
11         input = Console.ReadLine();
12         position = 0;
13
14         Console.WriteLine("\nInput          Action");
15         Console.WriteLine("-----");
16
17         if (S() && position == input.Length - 1)
18             Console.WriteLine("-----\nString is successfully parsed.");
19         else
20             Console.WriteLine("-----\nError in parsing string.");
21     }
22
23     static bool S()
24     {
25         Console.WriteLine("{0,-15} S -> X$", input[position]);
26         if (X() && position == input.Length - 1 && input[position] == '$')
27             return true;
28         return false;
29     }
30
31     static bool X()
32     {
33         Console.WriteLine("{0,-15} X -> Y X'", input[position]);
34         if (Y() && X_())
35             return true;
36         return false;
37     }
38
39     static bool X_()
40     {
41         if (input[position] == '$')
42         {
43             Console.WriteLine("{0,-15} X' -> % Y X'", input[position]);
44             position++;
45             if (Y() && X_())
46                 return true;
47             return false;
48         }
49         else
50         {
51             Console.WriteLine("{0,-15} X' -> e", input[position]);
52             return true; // e production
53         }
54     }
55
56     static bool Y()
57     {
58         Console.WriteLine("{0,-15} Y -> Z Y'", input[position]);
59         if (Z() && Y_())
60             return true;
61         return false;
62     }
63
64     static bool Y_()
65     {
66         Console.WriteLine("{0,-15} Y' -> & Z Y'", input[position]);
67         position++;
68         if (Y_() && Y_())
69             return true;
70         return false;
71     }
72 }
```

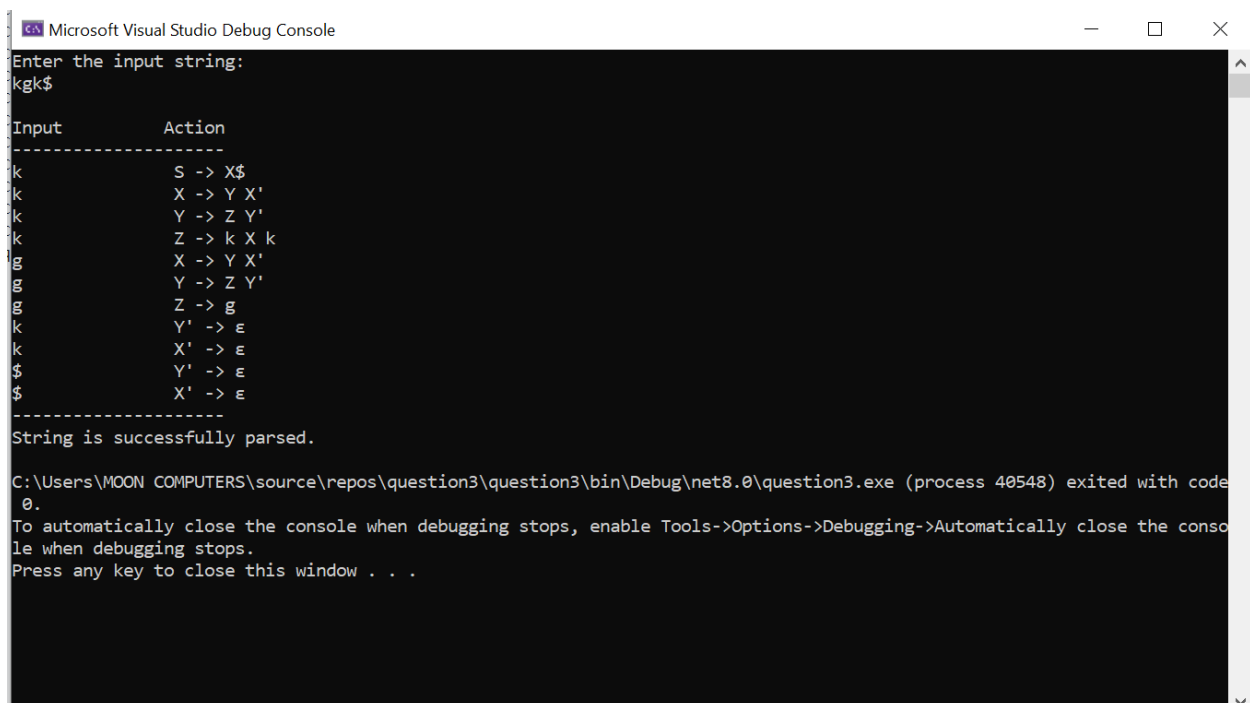


```
32 static bool X()
33 {
34     Console.WriteLine("{0,-15} X -> Y X'", input[position]);
35     if (Y() && X_())
36         return true;
37     return false;
38 }
39
40 static bool X_()
41 {
42     if (input[position] == '$')
43     {
44         Console.WriteLine("{0,-15} X' -> % Y X'", input[position]);
45         position++;
46         if (Y() && X_())
47             return true;
48         return false;
49     }
50     else
51     {
52         Console.WriteLine("{0,-15} X' -> e", input[position]);
53         return true; // e production
54     }
55 }
56
57 static bool Y()
58 {
59     Console.WriteLine("{0,-15} Y -> Z Y'", input[position]);
60     if (Z() && Y_())
61         return true;
62     return false;
63 }
64
65 static bool Y_()
66 {
67     Console.WriteLine("{0,-15} Y' -> & Z Y'", input[position]);
68     position++;
69     if (Y_() && Y_())
70         return true;
71     return false;
72 }
73 }
```



## Output:

Here is attached screenshot of both successful or unsuccessful parsing.



```
Microsoft Visual Studio Debug Console
Enter the input string:
kkk$

Input      Action
-----
k          S -> X$
k          X -> Y X'
k          Y -> Z Y'
k          Z -> k X k
k          X -> Y X'
k          Y -> Z Y'
k          Z -> k X k
k          X -> Y X'
k          Y -> Z Y'
k          Z -> k X k
$          X -> Y X'
$          Y -> Z Y'

Error in parsing string.

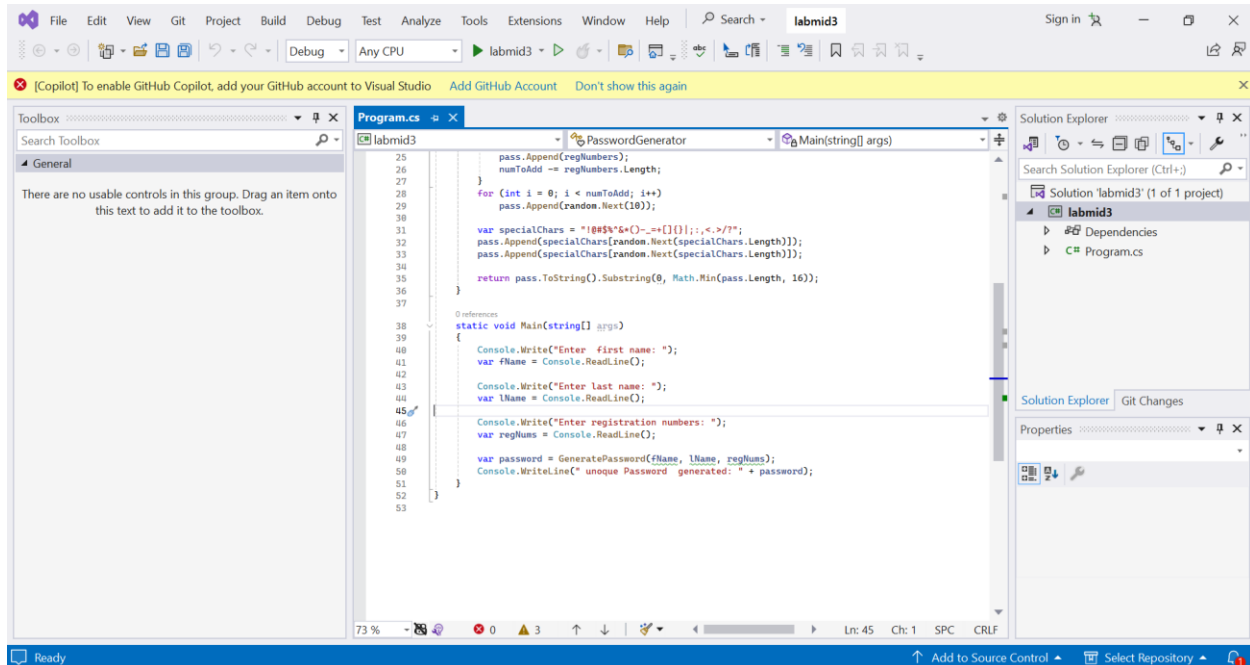
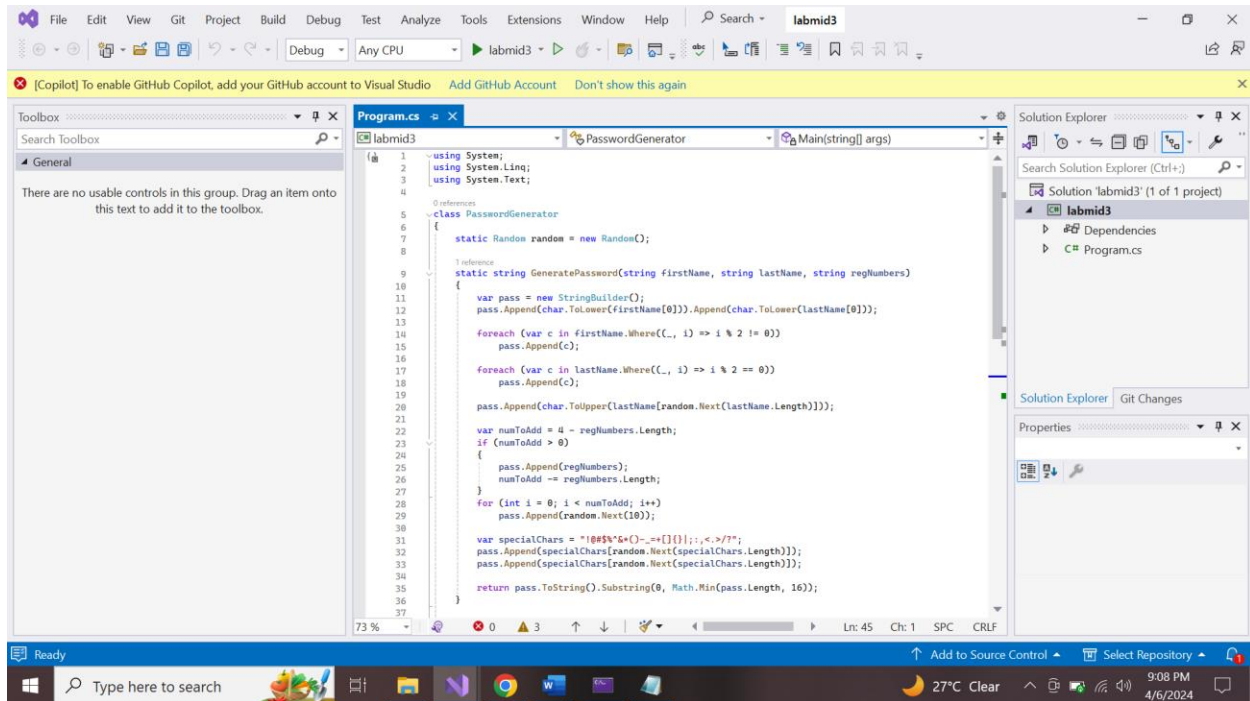
C:\Users\MOON COMPUTERS\source\repos\question3\question3\bin\Debug\net8.0\question3.exe (process 47868) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

### Question #03:

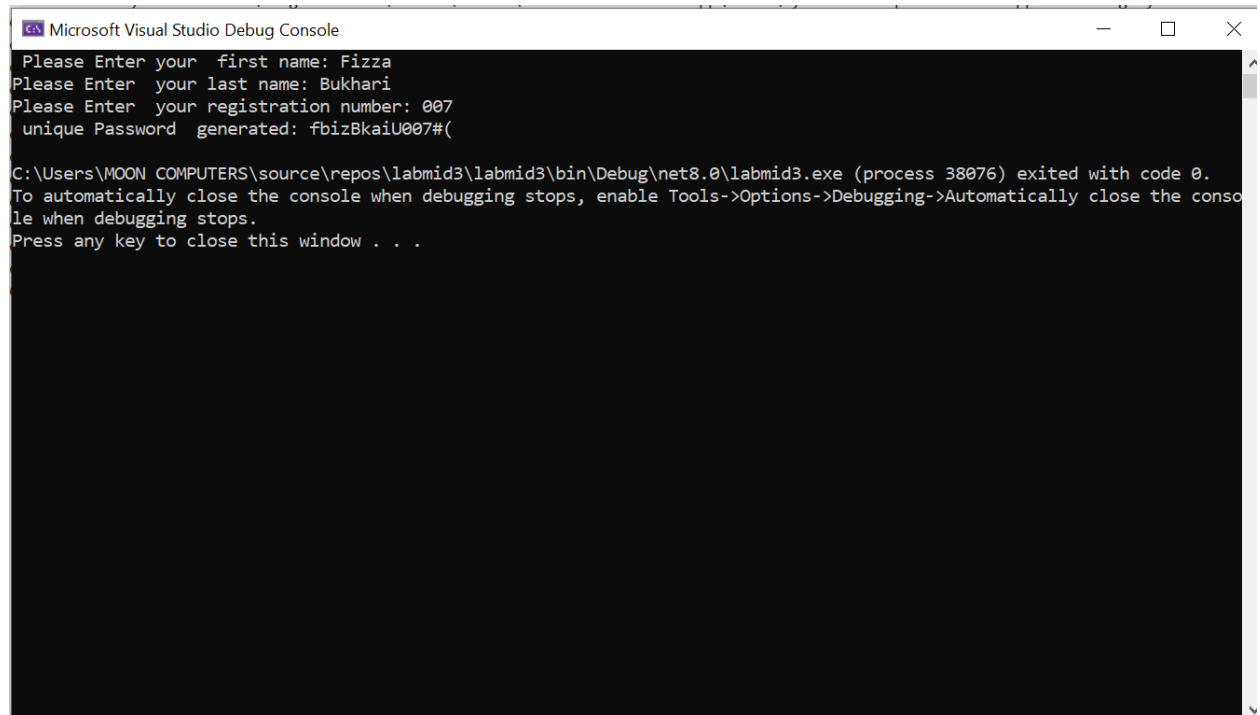
**Make a Password generator according the following rules:**

- a) At least one uppercase alphabet**
- b) At least 4 numbers, two numbers must be your registration numbers**
- c) At least 2 special characters**
- d) Must contain initials of first and last name**
- e) Must contain all odd letters of your first name.**
- f) Must contain all even letters of your last name.**
- g) maximum length of 16**

Now given below is the code to solve above problem in c#.



## Output:



A screenshot of the Microsoft Visual Studio Debug Console window. The window has a title bar with the text "Microsoft Visual Studio Debug Console" and standard window controls (minimize, maximize, close). The console area is black with white text. The output shows a program that prompts for user input and generates a password. The text in the console is as follows:

```
Please Enter your first name: Fizza
Please Enter your last name: Bukhari
Please Enter your registration number: 007
unique Password generated: fbizBkaiU007#(

C:\Users\MOON COMPUTERS\source\repos\labmid3\labmid3\bin\Debug\net8.0\labmid3.exe (process 38076) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

---